行列式の高速な精度保証付き数値計算法(数値シミュレーションを支える応用数理)

AUTHOR(S):

荻田, 武史; 尾崎, 克久; 大石, 進一

CITATION:

荻田, 武史 ...[et al]. 行列式の高速な精度保証付き数値計算法(数値シミュレーションを支える応用数理). 数理解析研究所講究録 2007, 1573: 45-52

ISSUE DATE:

2007-11

URL:

http://hdl.handle.net/2433/81323

数理解析研究所講究録
第1573巻 2007年 45-52

行列式の高速な精度保証付き数値計算法

荻田 武史 *,† 尾崎 克久 † 大石 進一 †

* 科学技術振興機構 (JST) 戦略的創造研究推進事業 (CREST)
† 早稲田大学 理工学術院

1 はじめに

本報告では，$n \times n$ 実行列 A の行列式 $\det(A)$ に対する高速な精度保証付き数値計算法を提案する．n が2や3など小さい場合は，行列式の定義式どおり計算することも可能であるが，n が大きい場合には，その方法は現実的ではない．このような場合，A のLU分解を用いるのが普通である．

本報告では，LU分解を用いて行列式を計算した場合に，得られた数値解の精度保証および行列式の符号の保証を高速に実現するアルゴリズムを提案する．また，数値実験によって，従来方式との比較も行う．

2 LU分解による行列式の計算

行列式を浮動小数点演算で計算するときに，「何よりも良い方法 [2]」と言われているのは，(部分軸交換付き) LU分解を用いることである．

そこで，$PA = LU$ を満たすような A のLU分解が得られたとする．ただし，P は軸交換に伴う置換行列，L は単位下三角行列，U は上三角行列である．このとき，$\det(A)$ は以下のように計算できる：

$$\det(A) = \det(P^T LU) = \det(P)\det(U) \tag{1}$$

すなわち，$\det(P)$ は，LU分解における軸交換の回数であり，$\det(U)$ は U の対角要素の積であるため，容易に得ることができる．

もちろん，実際にはLU分解は丸め誤差を含むので，式 (1) の最初の等式は，近似的に成立することになる．

一方，$\det(A)$ の精度保証は，以下の議論により実行できることがわかる．まず，A が正則 ($\det(A) \neq 0$) であると仮定すると，正則な L, U が存在する．次に，X_L と X_U をそれぞれ L と U の近似逆行列とする．また，$B := X_L P A X_U$ とする．このとき，$\det(X_L) = 1$ であることから

$$\det(B) = \det(P)\det(A)\det(X_U)$$

となる．すなわち，$\det(X_U) \neq 0$ であれば

$$\det(A) = \frac{\det(P)\det(B)}{\det(X_U)} \tag{2}$$

が成立する．ここで，A が悪条件でなければ，B はほとんど単位行列に近くなる．それは，$\det(B) \approx 1$ かつ B が対角優位な行列となることを意味する．Gershgorin の定理から，B のすべての固有値の包含が得られ，その積から $\det(B)$ の包含を得ることができる．これは，式 (2) によって，$\det(A)$ の包含を得ることができることを意味する．

ここまでの議論を用いて，Rump [7] は以下のような行列式の精度保証を行う効率的な INTLAB [6] のアルゴリズムを示した[1]．

アルゴリズム 2.1 (Rump [7]) $\det(A)$ を包含するアルゴリズム

```
function s = vdet(A)
  [L,U,P] = lu(A);                                  % LU factorization
  P = sparse(P);
  I = speye(size(A));                               % I: identity matrix
  XL = I / L;                                       % Solve X*L = I
  XU = U \ I;                                       % Solve U*X = I
  B = XL*intval(P*A)*XU;
  c = mid(diag(B));
  r = mag(sum(B-diag(c),2));
  g = midrad(c,r);                                  % Gershgorin circle
  s = det(P)*prod(g)/prod(intval(diag(XU)));
```

注釈 2.2 このアルゴリズムの最後の行は簡単にオーバフローやアンダフローを起こしてしまう．これを防ぐためには，例えば，LINPACK で採用されているように，$d = f \cdot 2^e$, $(0 \leq |f| < 1)$ のように表現するのが良い．

3 提案する精度保証法

本節では，Rump によるアルゴリズム 2.1 よりも高速な行列式の精度保証法および符号の保証法を提案する．

まず，行列式の符号を保証するために用いる以下の補題を示す．

補題 3.1 (Brönnimann et al. [1]) F を $n \times n$ 行列とする．任意の行列ノルム $\|\cdot\|$ に対して，$\|F\| < 1$ であれば

$$\det(I + F) > 0$$

である．ただし，I は $n \times n$ 単位行列とする．

もし，A が正則であれば，以下のような E が存在する：

$$P(I + E)A = LU \tag{3}$$

すなわち

$$E = P^T LUA^{-1} - I \tag{4}$$

となる．式 (3) から

$$\det(P(I + E)A) = \det(LU)$$

[1]著者によって文献 [7] のアルゴリズムを Matlab 向けに多少効率化してある．

であるため

$$\det(P)\det(I+E)\det(A) = \det(L)\det(U)$$

が成り立つ．$\det(L) = 1$ および $\det(P) = \det(P)^{-1} = \pm 1$ から

$$\det(I+E)\det(A) = \det(P)\det(U) \tag{5}$$

となる．また

$$\mathrm{sign}(\det(I+E)) \cdot \mathrm{sign}(\det(A)) = \det(P) \cdot \mathrm{sign}(\det(U)) \tag{6}$$

が成り立つ．ここで，$\det(U)$（およびその符号）は，U の対角要素の積（およびその符号）によって計算することができる．

したがって，式 (5), (6) および補題 3.1 から，以下の定理を得る．

定理 3.2 A を正則な $n \times n$ 実行列とする．L, U, P をそれぞれ任意の正則な単位下三角行列，上三角行列，置換行列とする．$E := P^T LUA^{-1} - I$ および $d := \mathrm{diag}(U)$ を定義する．このとき，$\|E\| < 1$ であれば

$$\det(A) = \frac{\det(P)}{\det(I+E)} \prod_{i=1}^{n} d_i$$

および

$$\mathrm{sign}(\det(A)) = \det(P) \cdot \mathrm{sign}\left(\prod_{i=1}^{n} d_i\right) = \det(P) \cdot \prod_{i=1}^{n} \mathrm{sign}(d_i)$$

が成立する．

もし $\|E\| \ll 1$ であれば，$I+E$ は単位行列に近くなり，$\det(I+E) \approx 1$ となる．Gershgorin の定理から，$I+E$ のすべての固有値の積の包含，すなわち $\det(I+E)$ の包含を計算できる．具体的には，$e := (1, \ldots, 1)^T \in \mathbb{R}^n$，$r := |E|e$ とすると，$\|E\|_\infty = \max_i r_i$ であり，$r_1, r_2, \ldots, r_n$ はそれぞれ Gershgorin 円の半径となる．したがって，$\|E\|_\infty < 1$ のとき

$$0 < \prod_{i=1}^{n}(1 - r_i) \le \det(I+E) \le \prod_{i=1}^{n}(1 + r_i) \tag{7}$$

が成立する．また，簡単な計算から，$n\|E\|_\infty < 1$ のとき

$$1 - n\|E\|_\infty \le \det(I+E) \le \frac{1}{1 - n\|E\|_\infty} \tag{8}$$

が成立することも分かる．

以上の議論から，以下の定理を得る．

系 3.3 A を正則な $n \times n$ 実行列とする．P, E, d を定理 3.2 と同様に定義する．また

$$\alpha := \det(P) \prod_{i=1}^{n} d_i, \quad \beta := (1 - n\|E\|_\infty)^{\mathrm{sign}(\alpha)}$$

とおく．このとき，$\|E\|_\infty < 1$ であれば

$$\alpha\beta \le \det(A) \le \frac{\alpha}{\beta} \tag{9}$$

が成立する．

次節では，ベクトル r の上限を計算する方法について議論する．

3.1 Gershgorin 円の半径

式 (4) から

$$r = |E|e = |P^T(LU - PA)A^{-1}|e \leq \|A^{-1}\||LU - PA|e \tag{10}$$

が成り立つ．ここで，$\|A^{-1}\|_\infty$ と $|LU - PA|$ の見積もりは独立に実行できることに注意する．ただし，$\|A^{-1}\|_\infty$ の見積もりをする過程で $|LU - PA|$ の見積もりを再利用することも可能である．

$\|A^{-1}\|$ の上限を計算するための様々な手法が知られている．よく知られている方法は以下のようなものである [3].

$$\|A^{-1}\|_\infty \leq \frac{\|R\|_\infty}{1 - \|I - RA\|_\infty} \tag{11}$$

この中で，主に計算コストを要するのは $\|I - RA\|_\infty$ の見積もりである．これまでに以下のような方法が知られている．浮動小数点演算による四則演算をそれぞれ 1 flop として，必要な計算量も示しておく．

- 標準的な方法 [3, 5]: $\frac{16}{3}n^3$ flops (R の計算: $\frac{4}{3}n^3$ flops, $I - RA$ の包含: $4n^3$ flops)
- 高速な方法 [3]: $\frac{2}{3}n^3$
- その他の方法 [4]

ここでは，Oishi-Rump による高速な方法 [3] を紹介する．以下は，A の LU 分解が

```
[L,U,P] = lu(A);
```

のように与えられている場合に，$\|A^{-1}\|_\infty$ の上限を求める Matlab のプログラムである．

アルゴリズム 3.4 (Oishi-Rump [3]) $\|A^{-1}\|_\infty$ の上限を求めるアルゴリズム

```
function c = fastinvnorm(L,U)
  n = length(L);                          % dimension
  I = speye(n);
  XL = I / L;                             % Solve X*L = I
  XU = I / U;                             % Solve X*U = I
  Eps = 2^-53;                            % machine epsilon
  e = ones(n,1);
  setround(+1)                            % rounding upwards
  v1 = abs(XU)*(abs(XL)*(abs(L)*(abs(U)*e)));
  v2 = abs(XU)*(abs(U)*e);
  gam = n*Eps / -(n*Eps - 1);             % gam >= n*Eps/(1 - n*Eps)
  v3 = gam * (2*v1 + v2);
  alpha = norm(v3,inf);                   % upper bound for norm(I-R*A,inf)
  if alpha >= 1
    setround(0)                           % rounding to nearest
    error('A is too ill-conditioned.')
```

```
  end
  v4 = abs(XU)*(abs(XL)*e);
  c = norm(v4,inf) / -(alpha - 1);     % upper bound for norm(inv(A),inf)
  setround(0)
return
```

注釈 3.5 $\|A^{-1}\|$ の上限を計算する別の方法としては，A の最小特異値の下限 $\mu > 0$ を計算することが考えられる．この場合，$\|A^{-1}\|_2$ の上限を $1/\mu$ によって計算する．

Higham [2] によって，LU 分解の事前誤差評価

$$|LU - PA| \leq \gamma_n |L||U| \tag{12}$$

が示されている．これから

$$|LU - PA|e \leq \gamma_n |L||U|e \tag{13}$$

が得られ，この右辺は $\mathcal{O}(n^2)$ flops で計算できる．

注釈 3.6 事前誤差評価は多くの場合に過大評価となるので，$LU - PA$ を陽的に計算することも考えられる．その計算には $\mathcal{O}(n^3)$ flops 必要であり，三角行列同士の積 LU は，密行列同士の積と比べて，計算機上での最適化が簡単ではないが（BLAS にも，そのような関数は用意されていない），得られる誤差評価は式 (13) よりもシャープになる．

以下に，$\|A^{-1}\|$ の上限を計算する高速な方法 [3] と式 (13) を組み合わせた行列式の符号を保証する Matlab アルゴリズムを示す．ただし，このアルゴリズムでは，行列式のゼロ判定はできないことに注意する．

アルゴリズム 3.7 sign(det(A)) を求めるアルゴリズム

```
function s = fastsigndet(A)
  n = length(A);  Eps = 2^-53;
  [L,U,P] = lu(A);  P = sparse(P);
  detP = det(P);
  c = fastinvnorm(L,U);                % アルゴリズム 2
  setround(+1)
  gam = n*Eps / -(n*Eps - 1);
  v_lu = abs(L)*(abs(U)*ones(n,1));
  norm_E = gam * c * norm(v_lu,inf);  % upper bound for norm(E,inf)
  setround(0)
  if norm_E < 1
    s = detP * prod(sign(diag(U)));
  else
    error('A is too ill-conditioned.')
  end
return
```

8行目の abs(L)*(abs(U)*ones(n,1)) は fastinvnorm の中で既に計算しているので，その結果を再利用しても良いが，計算量は $\mathcal{O}(n^2)$ flops なので，実際の計算時間にはあまり影響しない．

同様に，行列式の値を保証する Matlab アルゴリズムを以下に示す（一部に INTLAB の関数も使用している）．

アルゴリズム 3.8 det(A) の包含を求めるアルゴリズム

```
function s = fastvdet(A)
  n = length(A);  Eps = 2^-53;
  [L,U,P] = lu(A);  P = sparse(P);
  detP = det(P);
  c = fastinvnorm(L,U);                    % アルゴリズム 2
  setround(+1)
  gam = n*Eps / -(n*Eps - 1);
  r = (gam * c) * (abs(L)*(abs(U)*ones(n,1)));
  norm_E = norm(r,inf);                    % upper bound for norm(E,inf)
  if norm_E < 1
    du = prod(1 + r);
    setround(-1)
    dl = prod(1 - r);
    setround(0)
    d = infsup(dl,du);                     % dl <= det(I+E) <= du
    s = detP * prod(intval(diag(U))) / d;
  else
    setround(0)
    error('A is too ill-conditioned.')
  end
return
```

実際には，prod(intval(diag(U))) の部分でオーバフローやアンダフローを起こさないように工夫するべきである．

行列が悪条件であったり，行列式の値がゼロのような場合は，上記のアルゴリズム 3.7, 3.8 は

```
A is too ill-conditioned.
```

とエラーメッセージを表示する．すなわち，問題が悪条件であるため結果を出すことができないことはあっても，間違った結果を出すことはない．

行列式のゼロ判定については，上記のアルゴリズムでは対応できないため，別のアプローチが必要となるが，本報告では議論の対象としない．

4 数値実験

本節では，提案する行列式の精度保証法（アルゴリズム 3.8: **fastvdet**）の性能評価を行う．数値実験には，Intel Pentium M (1.2GHz) を CPU に持つ計算機を用い，すべての

計算は Matlab R2006a 上で倍精度演算を用いて実行した.

行列式の精度保証をする Rump の方法（アルゴリズム 2.1: vdet）と計算時間および精度保証結果について比較する．また，式 (11) の右辺の評価に標準的な方法 [3, 5] を用い，$|LU - PA|$ を事前誤差評価を用いずに丸めモード制御演算 [3, 5] で

```
setround(-1)
Tl = L*U - P*A;
setround(+1)
Tu = L*U - P*A;
Tu = max(abs(Tl),abs(Tu));
```

のように陽的に評価するアルゴリズムを **robustvdet** として，比較対象に加える．計算時間については，参考のため，行列式の符号判定をするアルゴリズム 3.7 (**fastsigndet**) による結果も示す．精度保証結果については，得られた区間 $s = [\underline{s}, \overline{s}]$ に対し，相対精度の意味で $\mathrm{rad}(s)/\mathrm{mid}(s)$ を示す．ただし，$\mathrm{mid}(s) := (\overline{s} + \underline{s})/2$，$\mathrm{rad}(s) := (\overline{s} - \underline{s})/2$ である．

まず，$n = 100, 500, 1000, 2000$ について，A を $[-1, 1]$ に一様分布する擬似乱数を要素とするような行列とした場合の結果を表 1, 2 に示す．このとき，A の条件数は 10^2〜10^4 程度であった．括弧内の数値は，**vdet** に対して何倍高速であるかを表している．

表 1: 計算時間 (秒)

n	vdet	fastvdet	robustvdet	fastsigndet
100	0.39	0.10 (3.91)	0.11 (3.56)	0.01
500	3.03	1.03 (2.93)	2.06 (1.47)	0.54
1000	16.77	4.57 (3.67)	13.53 (1.24)	3.61
2000	123.36	28.11 (4.39)	98.68 (1.25)	26.45

表 2: 乱数行列における精度保証結果 ($\mathrm{rad}(s)/\mathrm{mid}(s)$)

n	vdet	fastvdet	robustvdet
100	2.9e−10	6.2e−07	3.1e−09
500	8.5e−08	3.6e−03	4.3e−06
1000	1.1e−06	1.3e−01	5.6e−05
2000	1.4e−05	1.0e+00	9.6e−04

この結果から，計算時間については，提案方式（**fastdet**）は Rump の方法（**vdet**）と比較して 2.5 倍〜4.5 倍程度高速であることが分かる．また，精度保証結果については，行列サイズ n が大きくなるにつれて，提案方式の結果が大きく悪化することがわかる．これは，提案方式では式 (10) の

$$r = |P^T(LU - PA)A^{-1}|e \leq \|A^{-1}\||LU - PA|e$$

の評価において，高速化のために

- 逆行列のノルム $\|A^{-1}\|$ に分離したこと
- 事前誤差評価によって $|LU - PA|$ の上限を見積もっていること

が，r を過大評価している主な原因である．一方，Rump の方式は，n がある程度大きくなっても適用可能であり，提案方式よりもロバストであると言える．

次に，$n = 100$ に固定し，A の条件数を 10^2 から 10^{14} まで変化させたときの精度保証結果を表 3 に示す．テスト行列の生成には，Higham [2] の **randsvd** 関数を用いた．

表 3: 様々な条件数における精度保証結果 $(\mathrm{rad}(s)/\mathrm{mid}(s))$, $n = 100$

cond(A)	**vdet**	**fastvdet**	**robustvdet**
10^2	4.0e−10	5.7e−07	4.0e−10
10^4	1.8e−08	2.0e−05	2.8e−08
10^6	1.2e−06	7.0e−04	1.9e−06
10^8	5.8e−05	2.1e−02	9.3e−05
10^{10}	1.5e−03	8.1e−01	9.3e−03
10^{12}	1.3e−01	failed	6.0e−01
10^{14}	1.1e+00	failed	failed

この結果から，**fastvdet** は条件数がある程度まで大きい場合でも適用可能であるが，保証された精度は **vdet** と比べて 10^3 程度悪いことが分かる．また，**robustvdet** の結果は，**vdet** とほぼ同等であると言える．

参考文献

[1] H. Brönnimann, C. Burnikel, S. Pion: Interval arithmetic yields efficient dynamic filters for computational geometry, Discrete Appl. Math., 109:1-2 (2001), 25–47.

[2] N. J. Higham: Accuracy and Stability of Numerical Algorithms, 2nd ed., SIAM, Philadelphia, PA, 2002.

[3] S. Oishi, S. M. Rump: Fast verification of solutions of matrix equations, Numer. Math., 90:4 (2002), 755–773.

[4] K. Ozaki, T. Ogita, S. Oishi: Adaptive verification method for dense linear systems, Proc. NOLTA2006, 2006, 323–326.

[5] S. M. Rump: Fast and parallel interval arithmetic, BIT, 39:3 (1999), 534–554.

[6] S. M. Rump: INTLAB - INTerval LABoratory, Developments in Reliable Computing (T. Csendes ed.), Kluwer Academic Publishers, Dordrecht, 1999, 77–104.

[7] S. M. Rump: Computer-assisted Poofs and Self-Validating Methods, Handbook on Acuracy and Reliability in Scientific Computation (B. Einarsson ed.), SIAM, 195–240, 2005.